WHY IS THIS COURSE PUSHING FUNCTIONAL PROGRAMMING? – EDUCATING WELL-ROUNDED WEB DEVELOPERS WITH FUNCTIONAL JAVASCRIPT

M. Nurminen *
Tampere University
Tampere, Finland
ORCID 0000-0001-7609-8348
P. Niemelä
Tampere University
Tampere, Finland
ORCID 0000-0002-8673-9089
H.-M. Järvinen
Tampere University
Tampere, Finland
ORCID 0000-0003-0047-2051

Conference Key areas: Open and Online Teaching and Learning, Integrated learning environments for the digital native learners

Keywords: Functional programming, online course, web development

ABSTRACT

Imperative, object-oriented, and multi-paradigm programming languages are dominant in higher education. However, the use of functional languages is emerging. In parallel, features supporting functional paradigm (FP) have been added to languages traditionally categorized to other paradigms. Students benefit from fluency with several paradigms. In the studied primary Web Development course, JavaScript was used to familiarize students with selected features of the FP. The grading of the FP exercises was automatic. The automatic graders guaranteed the uniformity of feedback, treating each student's submissions equally. Exercise graders accepted multiple submissions, and their feedback suggested code improvements to students. After each of the ten exercise modules, students (N=257) estimated the topic difficulty and gave feedback. The post-module questionnaires emphasized FP topics in particular. The results show that students are aware of programming paradigms, but more support should be offered when learning new ones, for instance, having more concrete instructions and hands-on videos. The need for more instructions was apparent as, after the course's FP introduction, some students were still easily confused about such abstract FP concepts

*Corresponding author
M. Nurminen
mikko.nurminen@tuni.fi

as ‘functions as first-class citizens’. However, exercise results showed that students learned to use the taught FP features. They found them difficult, but for example, the JavaScript concurrency model was found to be more difficult.

1 INTRODUCTION

Paradigmatic classification of programming languages does not fully capture their multi-paradigm nature if a multi-paradigm language is classified under one paradigm. Yet paradigmatic classification provides a means to structure a vastly heterogeneous space of software design and implementation methodologies and their associated programming languages [1, 2]. General-purpose programming languages have adopted concepts that were first introduced in functional programming languages: for example lambda functions and new immutable data structures have been introduced to C++ and Java [3–6]. In addition, Python and JavaScript (ECMAScript version 6) support functions as arguments, and currying. These modern multi-paradigm languages can be used to introduce FP features to students. As programming languages are becoming multi-paradigm in increasing quantities, students should be taught about applying these paradigms.

JavaScript is used in the industry for implementing Web applications. Learning this language which is appreciated by potential employers adds to many students’ motivation. As a side dish for the main course of learning JavaScript, its functional features can be used to teach functional programming, too.

The context for this work is a basic Web Development course. The course’s intended learning outcomes would see students be able to design and implement basic Web server and client applications, and be able to describe and use FP feature presented during the course. Based on its use in the industry, and its FP features JavaScript was chosen as the programming language for the course. The course’s JavaScript exercises included FP tasks. In these FP tasks students designed and implemented code, approaching the task using the FP paradigm.

At the start of the course students’ understanding of functional programming concepts was collected with a questionnaire in order to come up with a suitable curriculum along with fitting learning activities. While the transfer of FP concepts was measured with the exercises, their retention was captured by a questionnaire aimed at checking the knowledge they have gained during the course.

Vast majority of the course’s online programming exercises were automatically graded, a couple used peer-reviews to introduce students to other students’ code and giving useful feedback. Students worked on their code in their own Git repositories. Upon student submission automatic grader programs wfould clone the students code, run it against the grader’s test code. After the tests were run, the grader would give students the points for the exercises, and importantly feedback on how a student could improve the code they had submitted. For each of the exercises, students had a change to submit their code multiple times, typically 20 submissions per exercise. This number of submissions could enable students to use the grader to iteratively improve their code, as they reflected on the features discussed in the exercise.

2 RELATED WORK

In the functional programming paradigm, functions are pure: pure functions depend

only on their input values, their parameters. Whenever a pure function would receive the same inputs, its output would consistently remain the same. Pure functions cause no side effects, such as state changes, where the state is defined as a Cartesian product of the values of all the variables of the program.

In mathematics, functions are also pure. Algebra is the domain of mathematics that is the most concerned with functions and variables. The transfer between algebraic constructs and computer science has been found to have favorable effects on learning in both directions [7, 8]. In their seminal work, 'How to design programs?' Felleisen et al. set guidelines for implementing a program in a reusable and secure manner, the key feature being purity [9]. Moreover, Design Recipe systematizes problem solving: a problem is divided into smaller solvable steps, i.e., functions, with a test-driven approach [9]. The use of Design Recipe has proven to foster the right order of operations and the composition of nested functions. Thus, Felleisen and Krishnamurthi state that functional programming provides the strongest evidence for the favorable effects on math skills [10].

Felleisen and Krishnamurthi list FP's advantages, i.e., more disciplined approach to problem solving, no side-effects, and data immutability. These features provide chances for applying mathematical structures to computer science, which is likely to appeal to academics and educators in CS field. A stricter FP approach would also foster code-level testability, security, an increased support for distributed and parallel computing, and large-scale development. The value of this approach is understood in the industry, too.

When looking at the popularity of functional programming languages in the TIOBE index[11], currently (Mar/2021) the first functional programming language is R in the 13th position, while and MATLAB and functional-flavoured Swift place 18th and 19th, respectively. However, some useful features of FP have been adopted by languages traditionally seen as representatives of other paradigms. These features include lambdas and some monadic structures. Lambdas have been introduced in mainstream languages such as C# (C# v2.0, 2006), PHP (PHP 5.3.0, 2009), C++ (version 11, 2011), and Java (version 8, 2014), whereas in JavaScript lambdas are inherently built-in to the structure of the language, thereby existent from the very beginning.

JavaScript has borrowed such FP features from functional Scheme, Scheme being one of the primary influencers. Thus, JavaScript enables demonstrating ideas from FP. Students are partly motivated by desire to optimize their skill set for transition to working life. For the teaching of FP to bear more fruit, we should seek ways to align teaching with the intentional motivation of students to be easily employable [12] and ways for lowering the threshold of learning [13].

2.1 JavaScript and functional programming

JavaScript has several features which enable functional programming. One of them is functions as first-class citizens: functions are accepted as variables, and as parameters of other functions. Moreover, JavaScript has a single-threaded, event-driven concurrency model. This enables concurrently executing for example user-initiated events, network requests, UI rendering, and animations. When developing Web applications with JavaScript asynchronous processing is a key feature, as there will be delays in the communication between clients and servers. The concurrency model relies on asynchronous callbacks and functions as parameters, i.e., the affordances of functional programming.

Higher-order functions can be used in JavaScript with, for example, Array's methods *map()*, *filter()*, and *reduce()*. ECMAScript version 6 and a great deal of libraries in JavaScript ecosystem are to some extent based on the ideas of FP.

However, for the majority of students, the move towards FP has proven to be quite a challenge [14]. This study then aims at improving the comprehension of these difficulties, and, ultimately, lowering the learning threshold. Thus, this study asks:

- RQ1: Which programming paradigms were students aware of before WebDev1 course?
- RQ2: Which JavaScript topics, and in particular which FP concepts, were students struggling with?
- RQ3: What could the course personnel do to make those FP concepts easier to grasp?

3 RESEARCH CONTEXT

The studied WebDev1 course is a comprehensive introduction to both front-end and back-end web technologies. Front-end technologies comprise of HTML5, CSS, and JavaScript, whereas back-end introduces Node.js. Unlike previous years, the utilization of Node.js frameworks, such as Express and Handlebars, was omitted. Instead, vanilla JavaScript approach was used primarily for pedagogical reasons: frameworks come and go, but HTTP and generic client-server architecture will stay. The course is targeted to third- and fourth-year students. The prerequisites for this course include three basic programming courses, and a basic database course. Prerequisites imply that course participants should have a considerable amount of programming routine, including a basic understanding of project work, e.g., from using Agile project management.

The study was conducted during the global COVID restrictions, where moving to remote teaching was a general recommendation. Thus, WebDev1 course replaced previous lectures with video recordings, and on-premises tutoring with online tutoring sessions. Students struggling with the exercises or the coursework assignment could get help during these so-called Kooditorio sessions, which were held in Teams. Kooditorio is a tutoring practice a-kin to primetime [15], except voluntary, where teachers and assistants answer questions, debug and co-implement students' code and scaffold them finalizing their exercises. Outside the set session times, the same Teams channels functioned as a Q&A discussion board. The students were encouraged to help each other and respond to these questions, cooperation between students was encouraged in course messages. During exercises and coursework assignment, the discussion was lively and the channel was extensively used. The assignment was done in pairs that, preferably, would also foster learning from each other. The earlier study has shown that earlier social connections primarily guide group formation, while help seeking within the groups is geared towards students with the most domain knowledge [16]. In this implementation, the groups were formed by course personnel with the help of an algorithm, which was designed to allocate pairs from students with the same target grades and performance level, also the responses to a group formation quiz were influential in the match making.

3.1 Grading

To complete the course, students had to pass weekly exercises, a coursework assignment, and an exam. The rounds overlapped so that the next exercise round opened before the current one closed. The topics consisted of, e.g., HTTP, Client-Server architecture, DOM, Web security, authentication, data persistence, and MVC architecture. In parallel, a transferral thread of FP was run.

The grading of exercises and the coursework assignment was automated where possible. Without automation, the amount of work would have been enormous, the theoretical maximum total number of submissions was 205.600. Course personnel of three could not have assessed this number of submissions manually.

3.2 FP topics in the course content

The imperative paradigm is dominant in the curricula of CS students. This basic Web Development 1 course (hereafter: WebDev1) might be the first time they are exposed to FP. For learning functional programming, WebDev1 course sought to act as an easy starting point: during the course the following aspects of FP were discussed:

- the emphasis on “no side effects” and immutability (*const* rather than *let* or *var*, *map()* rather than in-place changes with *for* loop)
- functions as first-class citizens
- higher-order functions (*Array* methods: *map()*, *filter()*, and *reduce()*)
- arrow functions.

Listed topics were covered more deeply in materials and exercises. They were complemented by a cursory introduction of the following in materials and questionnaires: recursion and higher-order functions in general, continuation-passing style, currying, functors / monads, and finally railway-oriented programming. Fifth exercise round had exercises on the central functional programming practice of avoiding side effects from function calls and the associated immutable data, as well as higher-order Array functions map,() filter(), and reduce(). The tenth exercise round included discussion in the materials on topics related to functional programming: higher-order functions, recursion, functors/monads, continuation-passing style, and railway-oriented programming.

In JavaScript, the prominence of functions manifests itself also in handling asynchronicity with callbacks; this in contrast to other imperative languages, where functions as parameters are not so common. The syntax for using asynchronicity in JavaScript has evolved in steps from callbacks to promises, and finally to async/await. Promises provide two-fold handling options: promise can either resolve or reject. Here again, functions play the main role, whereas async/await returns to a more conventional control flow. In the scale of one function, async/await behavior is sync-like: the magic happens in the background, where actions do not block the execution of each other. Asynchronous callbacks and high-order functions force students to practice coding in a function-oriented manner.

Many software developers start their careers in Web development. The JavaScript code that runs in the context of a web browser with the help of a library such as React, provides a learning laboratory for CS students and prepares them for future multi-paradigm challenges. A functional programming library such as Ramda can add a great deal of functional look and feel to showcase algorithm design in a more declarative and functional way. Ramda is suitable for demonstrating side-effect free algorithm design with

pure and curried functions.

JavaScript can be a good tool for learning and teaching the ideas of functional programming. Once the teaching of the language constructs is combined with teaching practices that are not only motivating, but also conceptually rewarding, we may be able to hit the sweet spot of designing learning solutions.

If JavaScript were taught along with libraries, such as React and Ramda, this might help the students to grasp the skills required by employers and make the learning curve for functional programming easier. The React library is considered moderately easy and therefore may be suitable for teaching [17]. Both libraries promote side-effect free style for writing software.

3.3 Method and research instruments

The WebDev1 course will be developed in iteration cycles twice a year. The development started in 2019 [18]; in 2020, it was continued by the introduction of new autograders mainly for static code analysis. Cyclic development with reflective redesign phases is characteristic of design-based research (DBR) [19] [20] [21]. DBR mandates a guiding background theory, and this study leans of the previous findings of flipped learning in the course arrangements [22–24]. On FP, we looked back to research reporting on courses that applied FP principles.

In DBR, educational solutions are combined with the empirical interventions and proof: DBR systematizes course development cycle of *design, development, enactment, and analysis* [25–27]. Here the *cycle* represents a course term. The retrospective analysis inserts requirements into the design of the next implementation [28–30]. The redesign implies *'reflective conversation with the situation'* [31], whereby course personnel observes the effects of new arrangements and refines them if necessary.

At the start of the course, the prior knowledge of students was captured using a pre-questionnaire. The questionnaire consisted of 20 Likert-scale questions followed by three open-ended questions about programming experience and knowledge. The Likert-scale questions corresponded with the topics of each exercise round, complemented with some additional transversal skills in functional programming. The three open-ended questions were:

- My programming experience in years.
- Programming languages that I know.
- Programming paradigms I am aware of.

Each exercise round was completed with a similar questionnaire collecting students' open questions, and difficulties with the taught topics.

4 RESULTS AND DISCUSSION

4.1 Students' awareness of programming paradigms

The data for RQ1 'Which programming paradigms were students aware of before the FP intervention of WebDev1 course?' was collected as a part of the pre-questionnaire. The mentioned programming paradigms are presented in Table 1. Students could list any number of paradigms they were aware of. 'Being aware' may have been too vague an expression. Some students listed as much as nine paradigms, it can be assumed that at least a few of them interpreted the purpose to be to list all the paradigms they

had even some familiarity with. On the other end of the spectrum, 36 students left the question empty, and additional 12 students answered with a variant of 'I do not know any paradigms'. Some students mixed programming languages with programming paradigms. When looking at this data, it should be kept in mind that the number of programming paradigms that exist is not a universally-agreed upon quantity. For example, procedural programming is a form of imperative programming, and there is an overlap between logical programming and declarative programming.

Paradigm	Mentions (n)	Paradigm	Mentions (n)
Object-oriented	162	Functional	114
Procedural	47	Imperative	34
Declarative	22	Event-based	10
Logical	8	Structured	6
Data-driven	2	Reactive	1

Table 1: Mentioned programming paradigms

As expected, the object-oriented paradigm was the most well-known, being mentioned by 162 students: first programming courses in Tampere University use object-oriented programming languages. In contrast, the imperative programming paradigm got only 34 mentions, even if the object-oriented paradigm can be categorized as an imperative paradigm.

Surprisingly, the functional paradigm got second-most mentions, with 114 students stating they are aware of it. This was unexpected, as there had not been any functional programming courses in their curriculum. Functional languages multi-paradigm languages popularity both in- and outside academia could be one valid explanation, with possible contribution by JavaScript. Functional paradigm may be grouped under declarative paradigms, which itself got 22 mentions.

4.2 Experienced difficulty levels of exercise topics

Students' answers to questionnaires in each exercise round about the experienced difficulty of the topics discussed used a range from difficult(1) to easy(5). The number of respondents dropped towards the end. The first topic in the first exercise round questionnaire (Git) received submissions from 227 students, while the last topic in the last questionnaire (Error handling) received 110. There is a noticeable drop in the number of students filling questionnaires when exercise rounds moved from ninth data persistence exercise round to the tenth which handled MVC and code quality.

Among topics that students felt were the easiest are Git version control (154 viewing the topic as easy or somewhat easy [67.8%, n=227]), JavaScript's events (125 [61%, n=205]), JSON file and data format (105 [57%, n=184]) and LocalStorage API used for data persistence in the browser(87 [55%, n=157]). These topics come from different exercise rounds, so the relative ease students felt with the subject matter is not explained by the one or two exercise rounds having a familiar subject matter. While many of the students studied will have encountered Git and JSON in their earlier courses, JavaScript events or LocalStorage API were not taught in any earlier course in Tampere University.

The most difficult general topics included the REST architecture (76 students regarded the topic difficult/somewhat difficult [36%, n=209]), spread operator (72 [39%n=186]), CORS (63 [36%, n=174]), sessions and streams (91 [52%, n=175]). Among asynchronous JavaScript topics especially difficult were Promises (83 [47%, n=177]) and async/await (88 [49%, n=178]). These two asynchronous topics are so fundamental,

that they can be seen as threshold concepts.

Looking at students' difficulties with FP topics relatively most difficult topics were perceived to be: the requirement for function calls to have no side effects (86 students reporting the topic was difficult or somewhat difficult [36%, n=180]) and the closely related immutability of the object's state data (73 [40%, n=183]). Other difficult FP topics were functors/monads (48 [43%, n=112]), railway oriented-programming (46 [41%, n=111]), and higher-order functions (40 [36%, n=112]).

These terms are part of the knowledge students will require to understand and apply functional programming in problem solving in the future. That these topics were harder to grasp comes as no surprise, if we keep in mind that the curriculum is such that they will take functional programming courses during the next years of their studies.

These answers can be viewed in the context of students' answers to the pre-questionnaire where 114 students reported being aware of FP. In the same questionnaire the reportedly difficult FP topics are at the same time central to the paradigm. This can be interpreted to mean that students aware of FP, but were not introduced to FP in the previous courses. The gentle FP primer offered as part of this course was then well placed to find an audience that was already aware of the paradigm.

4.3 Peer tutoring and scaffolding

Help seeking during the course was enabled using Teams channels where students could ask and answer questions. 240 students participated in the discussions, 15 received extra points ranging from 2 to 5 for being active on the channels. Overall, the activity points correlated with higher course grades, as all but one of the 15 students received grade 3 or higher. As these students had answered questions in the channels, the channels provided peer-tutoring from more knowledgeable others [32] in complement to scaffold by the course personnel.

The opportunity for earning the extra activity points was made known to students at the start of the course. Pursuing those extra points might have been a part of what motivated the active students, but still their efforts enabled other students receive competent and timely help. If this would have happened without the extra activity points, is up to debate.

5 CONCLUSIONS

RQ1: Which programming paradigms were students aware of before the FP intervention of WebDev1 course? Several paradigms were mentioned, most frequently object-oriented and functional paradigms. Students' answers, however, revealed that many did not fully understand what a programming paradigm is. This can be seen as a product of the code-first approach in the earlier courses. Students can design and implement code without identifying the underlying paradigms and their pros and cons. The introduction of prominent paradigms would give them a basic structure for understanding and classifying programming languages.

RQ2: Which JavaScript topics, and in particular which FP concepts, were students struggling with? Asynchronous features of JavaScript, using Promises and async/await, were the most difficult topics, and encountering JavaScript concurrency model for the first time during this course added to the difficulty. Asynchronous features are extensively used in modern web development, thus their perception is of pivotal im-

portance. Compared with async, FP was considered easier: FP topics were reported being difficult or only somewhat difficult. The difficult topics included, among others, no side-effects, and immutability.

RQ3: What could the course personnel do to make those FP concepts easier to grasp? The student feedback suggested that the course personnel should create more concrete examples covering both lecture slides and hands-on videos. Videos should be short and to-the-point, and material and attached exercises should flow in sync. From students' feedback we could also read that very abstract concepts, such as the ones borrowed from mathematics, should be properly primed and explained. These concepts comprise, e.g., higher-order functions and functions as first-class citizens. Using JavaScript libraries such as Underscore.js, Ramda and Lodash could be used for making understanding these concepts easier to see and implement at code level while using functional programming.

6 FURTHER STUDIES

The results direct the improvements of the course in the next DBR cycle, the main result being a call for a more concrete FP approach: examples and hands-on videos could be elevated with visualizations to demonstrate FP and async principles that were ranked the most challenging topic. The right rhythm of videos and exercises may be found with the help of flipped learning research.

Data collected by Learning management system XYZ and GitLab is massive and would provide material for learning analytics; the results should be accessible for both teachers and students. Students could be keen on performance comparisons, though this might induce unnecessary competition. Comparing students' performance with their own earlier performance is safer. Current Learning management system XYZ graders check code quality and conventions. In addition, a grader visualizing the learning process would be handy in improving students' consciousness of their strengths and weaknesses, preferably with suggestions of exercises to fill the gaps. The anticipated grader is called a self-reflection grader.

References

[1] Shriram Krishnamurthi and Kathi Fisler. *Programming Paradigms and Beyond*, page 377–413. Cambridge Handbooks in Psychology. Cambridge University Press, 2019. doi: 10.1017/9781108654555.014.

[2] Greg Michaelson. Programming paradigms and computational thinking. 2018.

[3] Jaakko Järvi and John Freeman. C++ lambda expressions and closures. *Science of Computer Programming*, 75(9):762–772, 2010.

[4] Zoltán Porkoláb. Immutables in c++: Language foundation for functional programming. In *Central European Functional Programming School*, pages 75–110. Springer, 2015.

[5] Venkat Subramaniam. *Functional programming in Java: harnessing the power of Java 8 Lambda expressions*. Pragmatic Bookshelf, 2014.

[6] Dean Wampler. *Functional Programming for Java Developers: Tools for Better Concurrency, Abstraction, and Agility*. O'Reilly Media, Inc., 2011.

[7] Emmanuel Tanenbaum Schanzer. *Algebraic Functions, Computer Programming,*

and the Challenge of Transfer. PhD thesis, Harvard University, the Graduate School of Education, 2015.

[8] Emmanuel Schanzer, Kathi Fisler, Shriram Krishnamurthi, and Matthias Felleisen. Transferring skills at solving word problems from computing to algebra through Bootstrap. In *Proceedings of the 46th ACM Technical symposium on computer science education*, pages 616–621. ACM, 2015.

[9] M. Felleisen, R. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs, Second Edition*. MIT-Press, 2014. URL http://www.ccs.neu.edu/home/matthias/HtDP2e/.

[10] Matthias Felleisen and Shriram Krishnamurthi. Viewpoint Why computer science doesn't matter. *Communications of the ACM*, 52(7):37–40, 2009.

[11] TIOBE. TIOBE Index for November 2020, 2020. URL https://www.tiobe.com/tiobe-index/.

[12] RICHARD M Ryan and Arlen C Moller. Competence as central, but not sufficient, for high-quality motivation. *Handbook of competence and motivation: Theory and application*, pages 216–238, 2017.

[13] Michael Grady. Functional programming using JavaScript and the HTML5 canvas element. *Journal of computing sciences in colleges*, 26(2):97–105, 2010.

[14] Chris Clack and Colin Myers. The dys-functional student. In *International Symposium on Functional Programming Languages in Education*, pages 289–309. Springer, 1995.

[15] Pekka Koskinen, Joni Lämsä, Jussi Maunuksela, Raija Hämäläinen, and Jouni Viiri. Primetime learning: collaborative and technology-enhanced studying with genuine teacher presence. *International journal of STEM education*, 5(1):20, 2018.

[16] Mikko Nurminen, Pietari Heino, and Petri Ihantola. Friends and gurus: Do students ask for help from those they know or those who would know. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, Koli Calling '17, page 80–87, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450353014. doi: 10.1145/3141880.3141905. URL https://doi.org/10.1145/3141880.3141905.

[17] Eric Wohlgethan. Supporting web development decisions by comparing three major javascript frameworks: Angular, react and vue.js. page 47, 2018. URL https://reposit.haw-hamburg.de/bitstream/20.500.12738/8417/1/BA_Wohlgethan_2176410.pdf.

[18] Pia Niemelä and Mikko Nurminen. Rate your mate for food for thought: Elsewhere use a grader. In *Proceedings of the 12th International Conference on Computer Supported Education - Volume 2: CSEDU,*, pages 422–429. INSTICC, SciTePress, 2020. ISBN 978-989-758-417-6. doi: 10.5220/0009564204220429.

[19] Paul Cobb, Jere Confrey, Andrea diSessa, Richard Lehrer, and Leona Schauble. Design experiments in educational research. *Educational Researcher*, 32(1):9–13, 2003. doi: 10.3102/0013189X032001009. URL https://doi.org/10.3102/0013189X032001009.

[20] Peter Reimann. *Design-based research*, pages 37–50. Methodological choice and design. Springer, 2011.

[21] Barbara J. Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. Identifying design principles for cs teacher ebooks through design-based research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 191–200, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4449-4. doi: $10.1145/2960310.2960335$. URL `http://doi.acm.org/10.1145/2960310.2960335`.

[22] Erkko Sointu, Laura Hirsto, Mari Murtonen, et al. Transforming higher education teaching and learning environments–introduction to the special issue. *International Journal of Learning, Teaching and Educational Research*, 18(13):1–6, 2019. ISSN 1694-2493.

[23] Laura Hirsto and S Väisänen. Exploring the experiences of flipped-learning in a large lecture course in teacher education. In *ECER–conference, Dublin, Ireland*, 2016.

[24] Pia Niemelä, Aulikki Hyrskykari, Timo Poranen, Heikki Hyyrö, and Juhani Linna. Flipped Learning With Peer Reviews in the Introductory CS Course. In *Assessment, Testing, and Measurement Strategies in Global Higher Education*, pages 35–58. IGI Global, 2020.

[25] Feng Wang and Michael J Hannafin. Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development*, 53(4):5–23, 2005.

[26] Terry Anderson and Julie Shattuck. Design-based research: A decade of progress in education research? *Educational researcher*, 41(1):16–25, 2012.

[27] Rikke Ørngreen. Reflections on design-based research. In *Human Work Interaction Design. Work Analysis and Interaction Design Methods for Pervasive and Smart Workplaces*, pages 20–38. Springer, 2015.

[28] The Design-Based Research Collective. Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher*, pages 5–8, 2003.

[29] Jan Van den Akker, Koeno Gravemeijer, and Susan McKenney. Introducing educational design research. In *Educational design research*, pages 15–19. Routledge, 2006.

[30] Allan Collins. Toward a design science of education. In *New directions in educational technology*, pages 15–22. Springer, 1992.

[31] Donald A Schön. Designing as reflective conversation with the materials of a design situation. *Knowledge-based systems*, 5(1):3–14, 1992.

[32] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard University Press, 1980.